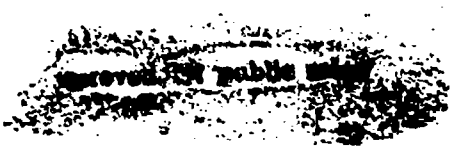
University of Southern California

PMoct: A Policy management tool for OCT based Design Systems for Multiple Domains

John Granacki and Tauseef Kazi

ISI/RR-93-387

October, 1993



INFORMATION SCIENCES INSTITUTE

310/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

UNIVERSITY OF SOUTHERN CALIFORNIA *INFORMATION SCIENCES INSTITUTE*
4676 Admiralty Way Marina del Rey, CA 90292

ISI Research Report
ISI/RR-93-387
October, 1993

PMoct: A Policy management tool for OCT based Design Systems for Multiple Domains

John Granacki and Tauseef Kazi

ISI/RR-93-387

October, 1993

University of Southern California
Information Science Institute
4676 Admiralty Way, Marina del Rey, CA 90292

Unclassified/Unlimited

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	6-13-94	Research Report

4. TITLE AND SUBTITLE
PMoct: A Policy management tool for OCT-based Design Systems for Multiple Domains.

5. FUNDING NUMBERS
J-FBI-91-282

6. AUTHOR(S)
John Granacki
Tauseef kazi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Specifying and maintaining the semantics of data used by CAD/CAE tools is often accomplished through written documentation. This documentation is used by the tool developers and is not usually computer processable which makes sharing of tools and libraries very difficult and cumbersome. In the OCT data management system developed at U.C. Berkeley, the semantics are referred to as the policy. In this paper, we describe a set of tools that has been developed to allow the user to specify and store the semantics (or policy) itself in the database, and ensure that new instances that are added to the database will be semantically consistent.

14. SUBJECT TERMS
CAD Framework
CAD Databases

15. NUMBER OF PAGES
6

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

PMoct: A Policy Management Tool for OCT-based Design Systems for Multiple Domains*

John Granacki and Tauseef Kazi
USC/Information Sciences Institute
4674 Admiralty Way
Marina del Rey, CA 90292

Abstract

Specifying and maintaining the semantics of data used by CAD/CAE tools is often accomplished through written documentation. This documentation is used by the tool developers and is not usually computer processable which makes sharing of tools and libraries very difficult and cumbersome. In the OCT data management system developed at U.C. Berkeley, the semantics are referred to as the policy. In this paper, we describe a set of tools that has been developed to allow the user to specify and store the semantics (or policy) itself in the database, and ensure that new instances that are added to the database will be semantically consistent.

1. Introduction and Motivation

OCT is a data manager for VLSI/CAD applications [1],[2] for storing information about various aspects of electronic design. OCT was designed to be as flexible as possible so that tool developers could use this to develop new tools, and accommodate different design methodologies and design flows. Often within a suite of tools like Lager [3], the developers have some consistent policies across tools; however, there is no "super policy" that a tool developer or user can consult. This is because the policy is hard-coded in the associated design management tool, in this case, DMoct [3], [4]. It is even more difficult to use tools and libraries that are not developed as part of the same suite of tools. To use tools from different suites requires translators to be developed and used to map the data from one policy to another. For example, using Lager policy and the octtools policy requires the *siv2sym* and *sym2siv* translators to allow designs produced using DMoct to use the TimberWolf tools[5]. This proliferation of policies does not foster the development of sharable libraries, primarily because DMoct would have to be modified or extended for new library policies such as for PCBs (printed circuit boards) or MCMs (multi-chip modules).

We are developing a system for board-level synthesis [6] and chose to use OCT to develop a sharable library of components[7]. We also had as our goal to be compatible with the system design tools and libraries that are part of the SIERA System developed at U.C. Berkeley. After developing a policy for board-level components in our library, we attempted to use the *SDL (Structure Description Language)* that is used to describe library cells and define the parameterized netlist. The SDL for an LM555 is shown in Figure 3. DMoct uses the SDL to create the SMV (*Structure Master View*), that is, the OCT view that stores the parameterized netlist. Part of the SMV for the LM555 is shown in Figure 2.

We found that DMoct had to be modified to allow the

* This work was funded by the Advanced Research Projects Agency under Contract number: J-FBI-91-282.

```
;;; LM555 Timer
(parent-cell LM555 (PackageClass PCB)
   (SIVMASTER LM555))
(parameters(PARTNAME "LM555")
   (PARTTYPE "ANALOG"))
(net Q ((parent (term Q (PINNUMBER '3)
   (TERMTYPE SIGNAL) (DIRECTION OUTPUT)))))
(net R ((parent (term R (PINNUMBER '4)
   (TERMTYPE SIGNAL) (DIRECTION INPUT)))))
(net TR ((parent (term TR (PINNUMBER '2)
   (TERMTYPE SIGNAL) (DIRECTION INPUT)))))
(net CV ((parent (term CV (PINNUMBER '5)
   (TERMTYPE SIGNAL) (DIRECTION INPUT)))))
(net THR ((parent (term THR (PINNUMBER '6)
   (TERMTYPE SIGNAL) (DIRECTION INPUT)))))
(net DIS ((parent (term DIS (PINNUMBER '7)
   (TERMTYPE SIGNAL) (DIRECTION OUTPUT)))))
(net VCC ((parent (term VCC (PINNUMBER '8)
   (TERMTYPE SUPPLY)))))
(net GND ((parent (term GND (PINNUMBER '1)
   (TERMTYPE GROUND)))))
(end-sdl)
```

Figure 1. SDL for LM555

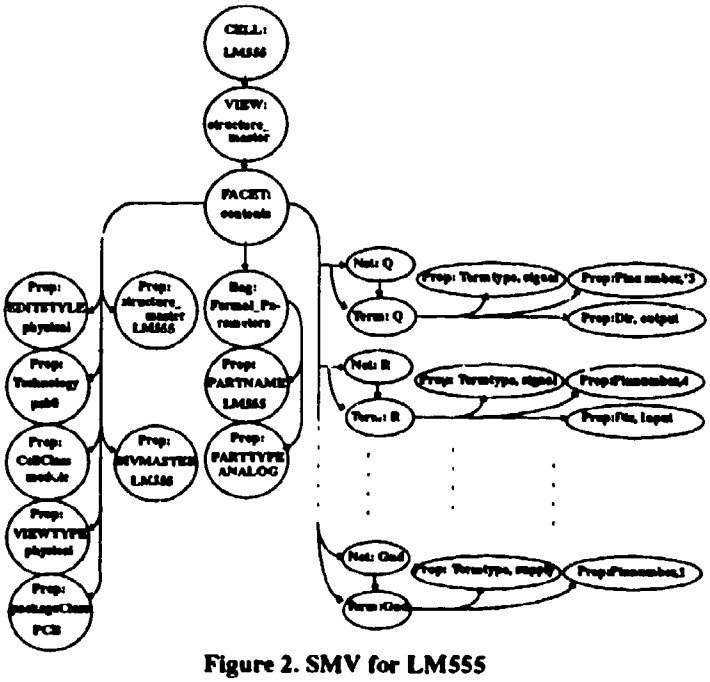


Figure 2. SMV for LM555

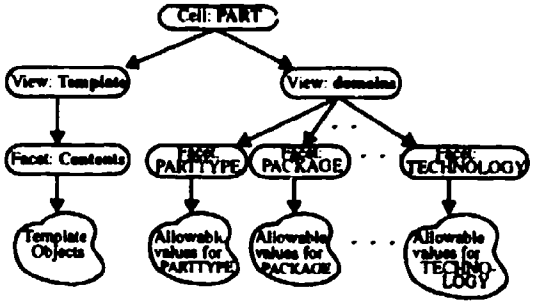


Figure 3. Generic PART Cell with Template View and Domains View

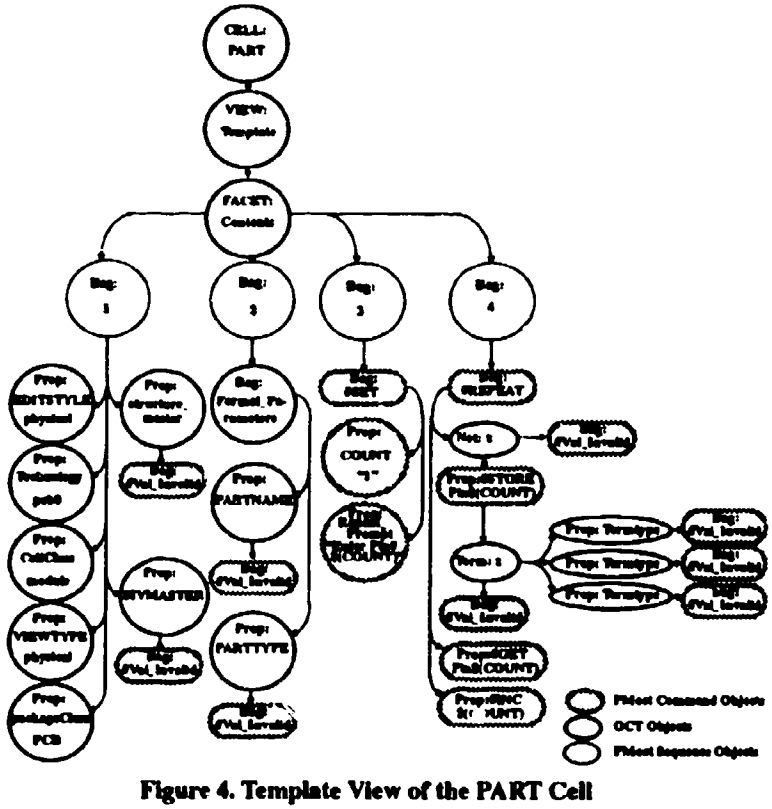


Figure 4. Template View of the PART Cell

description of the components for our board-level library. In fact, the Structure Description Language also had to be extended to handle some of the features required for representing components. Also, we were forced to use certain defaults consistent with the Lager policy --- these defaults were not necessary for any of our tools but their absence would not allow DMoct to successfully construct the SMV. For example, Lager policy requires a *layout-generator* be specified, our policy for a PCB does not require a layout-generator. Finally, the current Lager policy did not support some other OCT features that we required to make our components compatible with components in SIERA without storing redundant information. Redundant information not only requires more physical storage but it also contributes to maintenance difficulties. A change in the library policy would require that all occurrences of the redundant information be located and modified.

This motivated us to introduce the notion of a template and develop PMoct, a policy management tool.We created the *template view*, this view allows the user to specify the policy for a shareable library or a tool or suite of tools in OCT. The template view is used by PMoct to create library entries, as well as design instances, that is, the SMVs. All SMVs created from a template will be semantically consistent and will have the correct default properties and values as well as a consistent set of user specified values. If policy changes are necessary, only the information in the template view needs to be modified and PMoct will ensure that the policy changes are transparent to DMoct and other tools and across other libraries.

We give a brief review of OCT in Section 2. Section 3 describes the basic concepts incorporated in PMoct. Section 5 describes the use of PMoct in defining and storing the policy. In Section 6, an example of how PMoct enforces semantic consistency is presented. Finally, in Section 7, the benefits of PMoct are enumerated.

2. Overview of OCT

In OCT, the basic data element is a *cell*. A cell may have different *views*: for example, a component cell may have a physical view, a simulation view, a symbolic view, etc. Each view may contain one or more *facets*, which is the fundamental unit in OCT that is editable and exists as a physical file, while a cell and its views are directories.

The two most prevalent facets are the contents facet and the interface facet. The contents facet holds the actual data for the view and the interface facet gives the information that is accessible or necessary to other views.

A facet consists of a collection of objects which are related by *attachments* to one another. This enables the formation of a hierarchy of objects and a network of objects. For example, a net object may have a number of term objects attached to it. Similarly a box object may be attached to a term object and so on.

Since OCT does not have built in policies for the artifacts used in electronic design, it is up to the tool developer to give meaning to the data stored in OCT and hence implement and enforce the policy for a particular tool. We have created PMoct to assist the user in specifying and maintaining these policies.

3. PMoct: Basic Concepts

PMoct not only helps in maintaining a unified policy for a particular aspect of the design, but is also a single tool which can be used to handle the policies for different design methodologies and design flows. In addition to maintaining and creating policies, PMoct has the following capabilities that are not found in DMoct and SDL:

- PMoct reduces data redundancy by allowing a single instance of an object to be referenced by any number of other objects. This makes it unnecessary to find all the occurrences of an object when maintaining the library.
- PMoct has loop constructs to enable creation of multiple structures of the same type.
- PMoct is capable of getting values of objects from the user. It has constructs to specify default and user overrideable values in the template..
- PMoct has a mechanism to specify sets of allowable values for attributes.

4. PMoct: The Template

The template is an OCT cell having a name corresponding to the design artifact that it specifies. An example of the PART Template (cell) is shown in Figure 3. It has two views: the template view that contains a contents facet with the specification of the policy and the *domains view* which contains multiple facets each of which contains a set of allowable values for a particular OCT property. For example the PARTTYPE shown in Figure 2 is a property from the SIERA policy and may only take one of the following values: DIGITAL, ANALOG, RESISTOR, CAPACITOR and CONNECTOR. These are specified in the template by including a "PARTTYPE" facet under the domains view and putting all the valid part types in that facet.

5. Creating the Policy Using PMoct

As described in Section 3, the policy is defined as a contents facet under the template view as shown in Figure 4. The template cell may be created manually by the user using tools like attache or by running PMoct on a text script written by the user in *TDL* (*Template Description Language*). PMoct takes the TDL description as shown in Figure 5 and creates the template cell. Once the template cell is created, PMoct may then be used to create SMVs adhering to the policy defined in the template contents facet.

```
BAG:FORMAL_PARAMETERS {
  PRP:PartName {BAG:#VAL_INVALID}
  PRP:PartType{BAG:#VAL_INVALID }
  }
PRP:EDITSTYLE, physical
PRP:Technology, pcb0
PRP:Cell Class, Module
PRP:VIEWTYPE,physical
PRP:Package Class, PCB
PRP:structure_master{BAG:#VAL_INVALID}
PRP:SIV_master{BAG:#VAL_INVALID}
BAG:#SET{PRP:COUNT,"1"
  PRP:Repeat_Prompt,"Enter Pin" }
BAG:#REPEAT {
  NET:x{BAG:#VAL_INVALID
    PRP:#STORE,Pin$(COUNT) {
      TRM:x { BAG:#VAL_INVALID
        PRP:TermType {BAG:#VAL_INVALID}
        PRP:Pinnumber {BAG:#VAL_INVALID}
        PRP:Direction {BAG:#VAL_INVALID
  }}}}
  PRP:#GET,Pin$(COUNT) }
```

Figure 5. TDL for the PART Template

The template contains three kinds of objects: *OCT objects, command objects* and *sequence objects*. OCT objects are those which will actually exist in the final SMV either exactly as they appear in the template or with some of their values changed. Command objects are used to specify to PMoct how the OCT objects are to be processed. The command objects exist only in the template view and will not appear in the final SMV. Sequence objects are required to maintain an explicit sequence when processing the template because OCT does not guarantee any order for the retrieval of objects from a facet.

All objects in the template which are prefixed by a "#" symbol are command objects. These are read and acted upon by PMoct as they are encountered. The following subsections give a brief description of some of the commands objects.

When PMoct is run on the template, it processes all the objects attached to the sequence object with the value "1" first and then continues in integral order, that is, "2", "3", ... The sequence object "1" must be specified even for a single object and the numbering must be contiguous. This is done based on the order of statements in the TDL description and does not have to be explicitly declared.

5.1 Attaching an Object to Other Objects: #STORE and #GET

Existing design management tools such as DMoct do not provide a mechanism to arbitrarily attach an object to more than one other object. Attachments are implicitly hard-coded in DMoct, for example, in Figure 2 every terminal is attached to the facet as well as to the corresponding net. However, this cannot be specified explicitly in the SDL description shown in Figure 3 from which the SMV of Figure 2 was created.

By contrast, in PMoct, when an object must be attached to two or more objects, the object is cached by using the **#STORE** command. Then for every attachment, a

#GET command is used which simply gets the saved object and attaches it to the object specified. There is no restriction on how many other objects an object may be attached to.

5.2 Controlling Defaults and User Provided Values: #VAL_INVALID, #VAL_OVERRIDE, #ALLOWABLE_VALUES

All OCT objects that have the **#VAL_INVALID** command attached get their values from the user. For example, the property "PARTNAME" does not have a default value; therefore, a #VAL_INVALID is attached to the PARTNAME in the template as shown in Figure 4. When PMoct encounter this command it prompts the user for a value.

If there is no **#VAL_INVALID** command attached to an object, the value(s) associated with the object are used as the default value(s) and the object gets created with its value(s) being the same as the value(s) in the template. An example of this is the object labeled "Prop: Technology" in Figure 4 will get the value "pcb0."

If a **#VAL_OVERRIDE** command is attached to an object that does not have a **#VAL_INVALID** command attached to it, than the value it contains is taken as the default and the user is prompted to either accept the default value or enter a new one.

The **#ALLOWABLE_VALUES** command is used to restrict the value of an OCT property object to a domain specified in the domains view (as described in Section 4). This command object has a value associated with it. That value contains the name of the facet under the domains view which contains the set of allowable values. When PMoct encounters this command it will present a list of "allowable values" that can be selected by the user to complete the specification. This is a critical feature to ensure consistency across policies.

5.3 Creating Multiple Copies of a Set of Objects: #REPEAT

By using the **#REPEAT** command in PMoct a set of objects can be replicated multiple times. The set of objects to be replicated is simply attached to the **#REPEAT** command. When PMoct encounters the **#REPEAT** command, the user may create as many copies of the set of objects as required. This amounts to a significant reduction in the information that is specified and stored in the template. An example, of using this command is the algorithmic generation of pin numbers.

6. PMoct: An Example

When a policy is defined using PMoct, a level above the SMV is introduced. This level defined by the PMoct template formalizes the semantics of the data in terms of the domain. For example, our component library policy has an object REF-DES-PREFIX that comes from an enumerated set of values determined by IEEE Std. 315-1975. These values are associated with a REF-DES-CLASS object that is present in each component. The values in REF-DES-CLASS must match the values used to determine the REF-DES-PREFIX. For example an INTEGRATED_CIRCUIT_PACKAGE is assigned a "U" as a prefix and a CAPACITOR is assigned a "C". If the user does not use the same terminology, for example, if IC is used instead of the INTEGRATED_CIRCUIT_PACKAGE, then our physical design tools will not be able to find the correct prefix and would therefore report an error. However, if PMoct is used to instantiate a component SMV in the library, the user will be prompted to select the value from those defined for the REF-DES-CLASS Domain View. This ensures that only valid REF-DES-CLASS values will be associated with the REF-DES-CLASS object for a component. It also localizes the data used to determine the REF-DES-PREFIX with a REF-DES-CLASS and new classes can be created or new standards introduced in a single place. Currently, tools like oct2rinf (a translator for producing Racal Redac Intermediate Form data from a SIERA OCT database) hard-code this information in the program. When we tried to produce a similar tool for another design system, we had to duplicate the code that assigned the REF-DES-PREFIX and therefore proliferated this information. If changes or additions had to be made, they would now have to be carried out in several places.

Since PMoct can guarantee that values used to create an SMV from a template are consistent with a set of "allowable values" ---- these "allowable values" define the semantics for the values in a particular domain.

7. Conclusions

The principal advantages of storing the semantics (policy) in the database using PMoct tool described in this paper are:

- A single version of PMoct can be used to manage data and design flow across different domains of electronic design (for example, integrated circuits, printed circuit boards and multichip modules).
- PMoct makes it possible to quickly develop and integrate new tools into existing frameworks and more importantly to share tools and libraries that exist in different OCT-based design frameworks without the need for translators.
- Design instances created by tools other than PMoct can be checked for semantic consistency.
- Translation to and from non OCT-based systems can now be based on the information in the templates and translators can be specified and generated automatically.
- The policy is easily modifiable and changes can be propagated automatically throughout the library.

Although PMoct was developed using OCT for the underlying data management system, the ideas could be implemented in other CAD systems and with other database management systems.

Also PMoct would facilitate creating a CFI (CAD Framework Initiative) compliant interface to the component library and make sharing library data among other CFI compliant CAD systems trivial.

References

[1] David S. Harrison, Peter Moore, Rick L. Spickelmier, A. R. Newton, "Data Placement and Graphics Editing in the Berkeley Design Environment," *The proceedings of the IEEE International Conference on Computer -Aided Design*, pp. 24-27, Nov,1986.

[2] Rick Spickelmier and Brian C. Richards, "The OCT data manager," in *Anatomy of a Silicon Compiler*, Robert W. Brodersen, eds., pp. 11-24, 1992.

[3] C. Bernard Shung, et al, "An Integrated CAD System for Algorithmic-Specific IC Design," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 4, pp. 447-463, April, 1991.

[4] Brian C. Richards, "Design Management," in *Anatomy of a Silicon Compiler*, Robert W. Brodersen, eds., pp. 46-56, 1992.

[5] Carl Sechen, Kai-Win Lee, Bill Swartz, Mindy Lee and Dahe Chen, "The TimberWolf Standard Cell Placement and Global Routing Program," User's Guide for Version 4.2c, Yale University, October, 1987.

[6] John J. Granacki, "Research in Information Science and Technology: Systems Assembly Core Research," Final Technical Report, USC/Information Sciences Institute, November, 1992.

[7] John J. Granacki, Zia Iqbal and Tauseef Kazi, "A Component Library Management System and Browser," ISI Technical Report, USC/ Information Sciences Institute, April 1993..